TREK-USER-0002

TREK

DATA TUTORIAL

August 2014

TABLE OF CONTENTS

FIGURES

TABLES

TABLE PAGE

1 Welcome

The Telescience Resource Kit (TReK) is a suite of software applications and libraries that can be used to monitor and control assets in space or on the ground.

This tutorial describes what data is and how it is handled in TReK.

1.1 System Requirements

Windows 7, Red Hat Enterprise Linux 6.x.

2 Technical Support

If you are having trouble installing the TReK software or using any of the TReK software applications, please try the following suggestions:

Read the corresponding material in the manual and/or on-line help.

Ensure that you are correctly following all instructions.

Checkout the TReK Web site at http://trek.msfc.nasa.gov/ for Frequently Asked Questions.

If you are still unable to resolve your difficulty, please contact us for technical assistance:

TReK Help Desk E-Mail, Phone & Fax:

E-Mail:	trek.help@nasa.gov
Telephone:	256-544-3521 (7:00 a.m. - 3:30 p.m. Central Time)
Fax:	256-544-9353

TReK Help Desk hours are 7:00 a.m. – 3:30 p.m. Central Time Monday through Friday. If you call the TReK Help Desk and you get a recording please leave a message and someone will return your call. E-mail is the preferred contact method for help. The e-mail message is automatically forwarded to the TReK developers and helps cut the response time.

3 Introduction

This tutorial describes what data is and how it is handled in TReK. It assumes that you have read the TReK Getting Started Guide (TREK-USER-0001) and understand the difference between the TReK Toolkit and TReK Desktop. An understanding of previous versions of TReK, while helpful, is not required.

Previous versions of TReK used the terms telemetry and commanding to refer to data sent from the spacecraft and from the ground respectively. The telemetry and command data was available to the user through an application programming interface (API). These types of APIs will still be available in TReK Desktop. However, TReK Toolkit also provides access to the underlying code used by TReK to supply information to these APIs.

The term data is meant to be somewhat abstract. It doesn't matter if something is telemetry or commanding at the bit and byte level. It's just data. Three simple concepts cover most of what data is in TReK: packets, parameters, and parameter collections.

Packets are most often the data that are sent from one system to another system. Commands and telemetry are just packets. Command data is packets that tell another system to do something. Telemetry data is packets that supply information about the system sending the packet.

Parameters are the individual data values that contain information about the state of the sending system or actions to be taken by the receiving system. Parameters have a value and are either placed in the outgoing data or pulled from the incoming data. TReK uses the terms "build" to describe placing parameters in a packet and "extract" to describe pulling parameters from a packet.

Parameters are grouped with related parameters into collections named parameter collections. Parameter collections are the basic building blocks of packets which are the data sent from one system to another. Figure 1 has four views of the same packet. The first row shows a packet as a single entity that could be sent between systems. The second row shows that the packet is composed of parameter collections and another packet. The third row shows that eventually a packet will break down into a series of parameter collections. The final row shows that all parameter collections are a series of parameters. Each row is a different view of the same data.

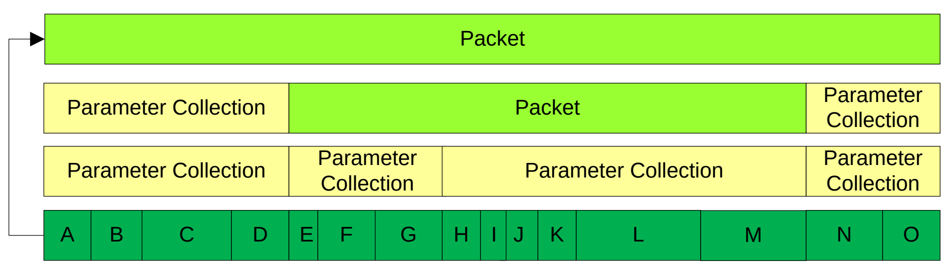

Figure 1 Packets, Parameter Collections, and Parameters

The sections that follow provide information on the concept of data in TReK. There is additional detail about using the Data API, including examples, available in online help when TReK is installed.

4 General Things

There are a few things that apply to most of things considered data in TReK. They are covered here.

4.1 Names

Almost everything associated with data gets a name. When you want TReK to give you a parameter's value, you need to know the parameter's name. The most used name will be the parameter's name, but other things will also have names. Everything that can be named can also have an alias. Where names should always be unique in a given context (more on that later), aliases do not have to be unique.

4.2 Descriptions

Anything that can be named can also have descriptions. TReK has three types of descriptions: short, long, and user. Short descriptions are usually just, well, short. Long descriptions often provide more information about a parameter. User descriptions are something that may be meaningful for a user and is intended to be reset by users. These distinctions really don't come into play except for TReK Desktop. For data, they are just three strings. There's no restriction on lengths (yes, a short description can be longer than a long description). However, when you use other capabilities in TReK such as databases, restrictions may be placed on the length of these descriptions.

4.3 Ownership

The owner is used to limit access to something within TReK. When you use the Data API directly, you have complete control of the data. However, in some environments such as commanding, you may need to restrict who has access.

5 Parameters

Parameters are the building blocks of data. Each parameter has a data type, length, location, number of samples, and other attributes. These attributes are used to place the parameter in a packet or pull the parameter out of a packet. The sections that follow will describe these attributes and introduce other details that will be covered in later sections.

5.1 Endianness

Before covering the attributes of parameters we'll cover an attribute of computer processors. The byte order of processors, or endianness, describes how the bytes are stored in memory. Computers are classified as big endian or little endian. Big endian computers store the most significant byte first. Little endian computers store the least significant byte first.

Knowledge of the endianness of the computer TReK is executing on is automatically obtained. However, the endianness of the computer sending or receiving the data must be supplied. The attribute that supplies that information is byte-order. In addition to little endian and big endian byte orders TReK also supports data that is byte swapped or word swapped. Figure 2 shows the byte order supported in TReK. Byte 0 is the most significant byte.

Big Endian:	0	1	2	3	4	5	6	7
Little Endian:	7	6	5	4	3	2	1	0
Byte Swapped:	1	0	3	2	5	4	7	6
Word Swapped:	2	3	0	1	6	7	4	5

Figure 2 Byte Order

5.2 Data Type

The data type allows TReK to know if the parameter represents a string, integer, floating point, or some other data. Most of the data types available in TReK originated from the Space Shuttle and International Space Station programs and are defined in the MSFC HOSC Telemetry Format Standard (MSFC-STD-1274 Volume 2). Additional data types have been added from other programs and from user requests.

For the purposes of this section data types will be grouped into three categories: numeric, byte-based, and time. Numeric data will mostly be floating point numbers and integers. Byte-based data will be data that is required to be on a byte boundary such as a string. Time data covers some of the different ways people have come up with over the years to represent time.

Each table that follows gives the name of the data type, length restrictions, and a description of the data. In some cases a reference may be made to other documents to provide a more detailed description of the data type. The allowed byte order for each data type is provided in the Data API documentation.

Data Type	**Length (bits)**	**Description**
Two’s Complement Integer	2-64	The native representation of signed integer
Unsigned Integer	1-64	An unsigned integer value
Binary Coded Decimal	4, 8, 12, or 16	An unsigned integer encoded as a binary coded decimal of 1-4 digits where each four bytes have a value of 0 to 9.
Distended Signed Integer	16 or 32	A 13-bit two’s complement integer represented as 16 or 32 bits. The sign bit always occupies the most significant

Data Type	Length (bits)	Description
		bit of the data and the other 12 bits occupy the least significant bits of data.
Sign and Magnitude Integer	2-32	A signed integer where the most significant bit represents the sign (0 – positive, 1 – negative) and the remaining bits represent the value.
IEEE Floating Point	32 or 64	IEEE 754: Standard for Binary Floating Point Arithmetic
Boolean	1	A single bit truth value.

Table 1 Numeric Data Types

Data Type	Length (bytes)	Description
NULL Terminated String	2-65,536	American Standard Codes for Information Interchange (ASCII) string that will have a NULL (hex zero) terminating character when building a packet.
Fixed Length String	1-65,536	ASCII string that may or may not have a NULL terminating character.
Unspecified Bytes	1-65,536	Binary data.

Table 2 Byte-Based Data Types

Data Type	Length (bytes)	Description
GPS Epoch Time	32	The number of seconds since 1980-01-06 00:00:00
EHS Time	48 or 52	Time represented as a series of fields indicating year, day, hour, minutes and seconds with an option fractional seconds. See MSFC-STD-1274 data types TEHS and TUDS for more information. *Note: TReK does not represent the status bits of TEHS as part of the time value, but as separate bit fields.*
ISS Time	40	GPS Epoch time followed by a single byte representing fractional seconds.
FASTSAT Time	48	GPS Epoch time followed by two bytes representing the number of milliseconds.
DEM Time	42	GPS Epoch time followed by 10 bits representing the number of milliseconds.
Unix Time	32	The number of seconds since 1970-01-01 00:00:00

Table 3 Time Data Types

5.3 Parameter Collections

Before continuing with attributes of parameters, a quick explanation of parameter collections is needed. A parameter collection is a group of parameters that are related. Parameters are referenced by their names in the collection, but their placement in the collection is based on a parameter's start bit.

5.4 Location and Samples

Each parameter will have a location in the parameter collection and a number of samples. The location in TReK is represented by the start bit relative to the beginning of the parameter collection containing the parameter.

Each parameter can have one or more samples of data. When a parameter has more than one sample an offset between the samples must be supplied. The number of samples will let TReK know how many samples to expect. Figure 3 shows an example of four parameters in a parameter collection and how their start location, length, parameter offset, and number of samples is used.

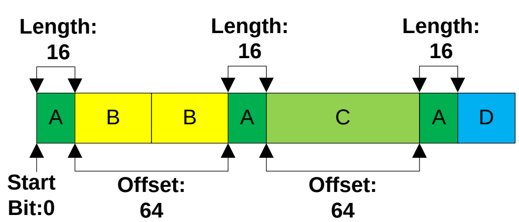

Figure 3 Parameter Location Attributes

The start location for a parameter collection begins at zero. Parameter A above is the first parameter in the collection and begins at start bit 0. It has a length of 16 bits. There are 3 samples of Parameter A and each sample is 64 bits apart (i.e., the number of bits from the end of one sample to the beginning of the next sample. Parameter B is 32 bits in length and has two samples with an offset of 0. Parameters C and D are both a single sample and do not use the offset attribute. They are 64 and 24 bits respectively.

5.5 Different Type of Parameter Values

Parameter values can be retrieved in different formats. The parameter's bit pattern as it appears in a packet is called the "raw" or "unprocessed" value. Often this representation is the same as the local data type for a computer. For example, two's complement integers are used for most system's representation of a signed integer. However, for cases where the representation of the parameter is different, you can still get to the unprocessed value if needed.

The most convenient way to get a parameter's value is as a type that is used on the computer you are currently using. These parameters are sometimes referred to as "converted" since TReK must translate the representation of data from the originating system to the local computer's representation. As this is the most requested form of data for TReK, we'll usually just refer to a parameter's converted value as its value throughout the documentation. The example that follows shows an integer that is encoded in the packet as a binary coded decimal. The raw value is how the data is represented in the packet. The converted value is how the value is represented on the computer processor as an unsigned integer. The decimal value is how a user sees the data when requesting a converted value.

Raw Value:	0100 0111 1001 0001
Converted Value:	0001 0010 1011 0111
Decimal Value:	4791

TReK can also transform a parameter's converted value to another value. This transformed value is referred to as a parameter's "calibrated" value. TReK has some built in calibrations, but is also capable of using code you write to perform unique calibrations for parameters as necessary. The details of calibration will be covered in a later section.

The final format for a value is the enumerated value. Enumeration is just the translation of an unsigned integer value to a string. For example, a single bit value may represent two states such as "On" and "Off". Enumerated values aren't necessarily used for programming decisions, but are great for displaying data values to users.

The availability of a parameter's value in each of these four forms is dependent on the data type and if other information is provided. The table below shows each of the data types in TReK and whether or not it is possible to retrieve the data as raw, converted, calibrated, or enumerated. For calibrated or enumerated values, more information is required. That will be covered later in the document.

Data Type	**Raw**	**Converted**	**Calibrated**	**Enumerated**
Two's Complement Integer	Yes	Yes	Yes	No
Unsigned Integer	Yes	Yes	Yes	Yes
Binary Coded Decimal	Yes	Yes	Yes	Yes
Distended Signed Integer	Yes	Yes	Yes	No
IEEE Floating Point	Yes	Yes	Yes	No
Boolean	Yes	Yes	No	Yes
NULL Terminated String	Yes	Yes	No	No
Fixed Length String	Yes	Yes	No	No
Unspecified Bytes	Yes	No	No	No
GPS Epoch Time	Yes	Yes	No	No
EHS Time	Yes	Yes	No	No
ISS Time	Yes	Yes	No	No
FASTSAT Time	Yes	Yes	No	No
DEM Time	Yes	Yes	No	No
Unix Time	Yes	Yes	No	No

Table 4 Value Availability for Each Data Type

5.6 Value Restrictions

Parameter values can have various restrictions placed on them. Sometimes this will be to prevent someone from setting an illegal value. It can also be to monitor a value for important information. These restrictions are explained below.

5.6.1 Ranges

A parameter can have low and high ranges associated with its value. When building a packet, you set each parameter's value. If a parameter has ranges defined, you will not be allowed to set a value outside of the allowed range. When extracting a packet, the parameter's value can be checked against the ranges to determine if someone else set the value outside of the allowed range.

5.6.2 Alarms

When retrieving values, the value can be checked for conditions that will trigger alarms. At the data level, alarms do not include any loud noise. Alarms are conditions that you've determined are important enough to know about. You will be informed about alarms via a parameter's status.

There are three types of alarms in TReK: low, high, and delta. Low alarms are triggered when a value is less than or equal to a low alarm point. High alarms are triggered when a value is greater than or equal to a high alarm point. Delta alarms are triggered when the value of a parameter changes at a rate as fast as the delta alarm point.

Figure 4 shows all of the low and high limits and their relationships. A value will only have one high or low alarm set.

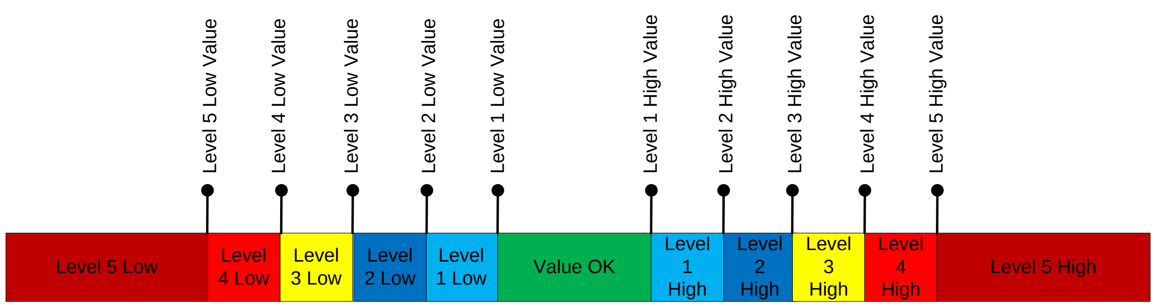

Figure 4 All High and Low Alarms

It is also possible to use only a subset of the high and low value alarms as shown in Figure 5.

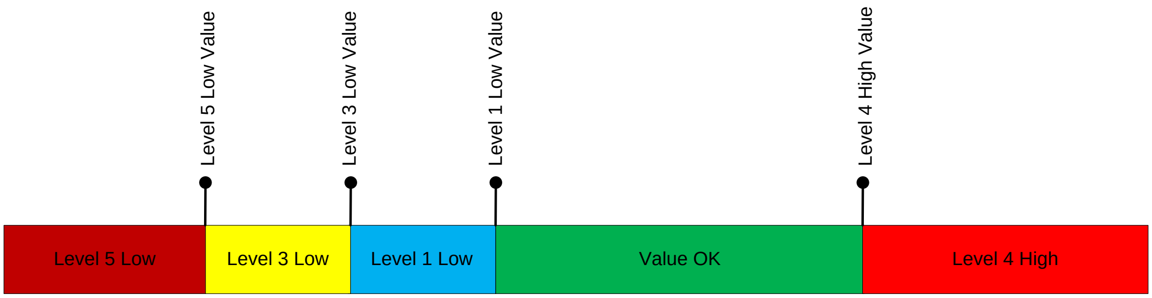

Figure 5 Selected High and Low Alarms

The delta alarm is triggered when the difference between consecutive values of a parameter is greater than the threshold set for the alarm. The threshold value is checked against the absolute value of the difference between consecutive parameter values. For example, consider a parameter with a delta alarm threshold of 10. The following series of values will show when a delta alarm is triggered:

45	No alarm, first value (i.e., no consecutive value to compare)
48	No alarm difference is only 3
60	Delta alarm triggered (60 – 48 = 12)
61	Ok
64	Ok
50	Delta alarm triggered (50 – 64 = -14)
56	Ok

Each alarm type (high, low, and delta) can have five levels which are referred to as Level 1, etc. Higher level numbers are considered more severe; a Level 5 alarm is more severe than a Level 1 alarm. Each of the five levels can be given a name. For example, you could name Level 2 “Caution” and Level 4 “Warning”.

You don’t have to use all five levels of any alarm or even all of the alarm types. Just choose the ones you need when a parameter’s value needs to be watched.

You also get to choose whether the converted or calibrated value is monitored for alarms.

5.7 Parameter Status

When TReK is extracting data additional information about a parameter’s value can be provided. The parameter status provides information on alarms that have been triggered, processing errors, etc.

5.7.1 Two Kinds of Status

There are actually two kinds of status provided: trek status and source status. The TReK status provides details on status that has occurred since TReK began processing the data.

When TReK detects an alarm limit has been reached or some other error, the TReK status is updated to include that information.

The source status is only available for data that was processed by other systems and tagged with a status. Most of the data you will process with TReK will not have a source status.

5.7.2 How Status is Returned

Status in TReK is returned as two 32-bit unsigned integers. One integer represents the TReK status and the other represents the source status. The TReK status is represented as bit fields with each bit representing a different type of status. A value of zero is considered no error. A value of one indicates that the bit represents some type of error. The source status is a 32-bit unsigned integer, but TReK only knows the value and not necessarily how each bit should be represented. These integers are good for programming decisions, but not great for user consumption.

These integer values can be converted to strings. The string returned will be of varying lengths depending on how many processing errors were detected. An empty string indicates that there are no errors. If the string isn't empty, TReK statuses are represented as an ASCII character. If there is additional status provided by the data source, it will be included at the end of the string and enclosed in a set of parentheses. There's an option to allow the source status to be represented as ASCII characters. If that option isn't selected, the source value will be displayed as an unsigned integer.

Table 1 shows the details for each status character available for TReK processing.   Status characters are listed in ASCII order.

Status Character	Definition
#	Level 2 delta limit error detected
$	Level 5 low limit error detected
&	Level 1 high limit error detected
*	Level 5 high limit error detected
+	Level 2 high limit error detected
-	Level 2 low limit error detected
0	Level 1 low limit error detected
?	Possible data loss detected
@	Level 1 delta limit error detected
A	Level 4 enumeration alarm detected
C	Conversion error detected
D	Level 4 delta alarm detected
E	Level 1 enumeration alarm detected
H	Level 4 high alarm detected
K	Calibration switch error detected
L	Level 4 low alarm detected
Q	Level 5 delta alarm detected

Status Character	Definition
R	High range error detected
T	Level 5 enumeration alarm detected
X	Alarm switch error detected
a	Level 3 enumeration alarm detected
c	General calibration error detected
d	Level 3 delta limit detected
e	Level 2 enumeration alarm detected
k	Checksum error detected (data quality suspect)
l	Bad length error detected
p	Processing error detected
r	Low range error detected
t	Illegal data type for calibration detected
v	Level 3 low limit detected
z	Packet length error detected
^	Level 3 high limit detected

Table 5 Status Characters

An example of what the status string would look like for a parameter where TReK detected both a Level 4 high alarm and a Level 1 delta alarm and the source status had a value of 15: "H@ (15)". In most cases you won't have more than one status character appear at any time.

6 Packets

Packets are the data that travel between systems. The packet is the largest aggregation of data in TReK.

6.1 Zones

Packets are divided into three zones: header, data, and trailer. One or more zones must be defined in a packet for it to be considered valid. Figure 6 shows each of the three zones and their relative locations. Each zone of a packet contains either another packet or a parameter collection.

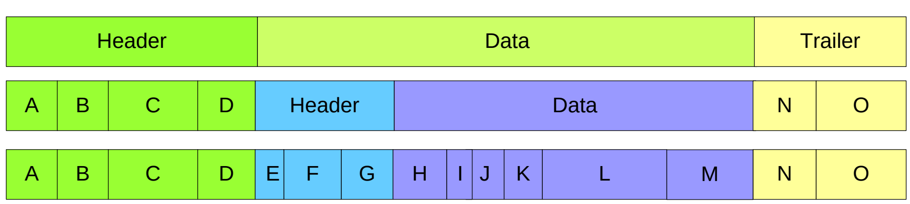

Figure 6 A Packet and Its Zones

The first line in the figure above shows a packet that has all three zones defined. The second level shows that the header and trailer zone are composed of parameter collections which contain one or more parameters. The data zone is composed of another packet which only has the header and data zones defined. The third line shows that the packet in the data zone of the top level packet is composed of two parameter collections and that all of the data in a packet will eventually break down into a series of parameters.

6.2 Attributes

There are five attributes that can be set for a packet in TReK: identifiers, counter, time stamp, length, and checksum. Each attribute can appear in any zone of the packet with the stipulation that the zone must contain a parameter collection and not a packet. Each attribute is optional, but some other features may not work if you don't define an attribute. For example, TReK uses the counter attribute to help determine if a delta error has occurred in a parameter. All of the attributes are parameters in the packet. An example is provided after all of the attributes are defined.

6.2.1 Identifiers

A packet can have one or more identifiers defined. A set of identifiers determine how a packet should be interpreted (i.e,. what parameters are contained in the packet). Most of the processing related to identification is found elsewhere in TReK. However, when building a packet specifying the identifiers allows you to guarantee the packet that is built will be correctly identified. The application process identifier (APID) in a CCSDS packet is an example of an identifier.

6.2.2 Counter

A packet can also have a parameter designated as the counter. When building the packet, TReK will automatically set the counter's value. The counter is also used when extracting data to determine if there was any missing data or if a delta error checking can be performed. Counters in packets are typically increasing values and reset to zero once the maximum has been reached. There are options to allow decrementing counters and different handling of minimum and maximum values, but they are rarely used.

6.2.3 Time Stamp

A time stamp can be designated for a packet. The time stamp parameter indicates the time the data was created. TReK will automatically set the time with the current system time when building packets if a time stamp parameter is defined for the packet. On extraction, you can retrieve the time parameter to determine when a packet was actually created.

6.2.4 Length

The length parameter of a packet is set by TReK when building data. This allows variable length packets to automatically have the correct length when sent.

6.2.5 Checksum

A checksum can be specified for a packet. TReK currently supports three checksum types (SUM16, CRC32, and MD5) which are described in the online help for the Data API. You can specify the checksum end points as the start of the packet, beginning of the data zone, the end of the data zone, and the end of the packet. Offsets are available to move the start and end points of the checksum as necessary.

When building a packet, the last parameter set is the checksum. TReK calculates the checksum based on the information provided when configuring the packet.

When extracting a packet, TReK will determine if the checksum in the packet matches what is calculated. If the checksum does not match, the data is still extracted. An extraction error is returned to indicate that the data is suspect and each parameter's status will have an error indicating a checksum error was detected.

6.2.6 Packet Attribute Example

Figure 7 shows how the packet from Figure 6 with each packet attribute type set. This packet has two identifiers and is sent every five seconds. Parameter M is variable length and will cause the packet length (Parameter D) to change each time.

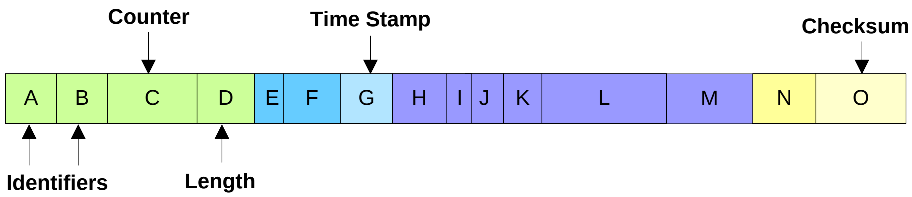

Figure 7 Packet Attributes

The two identifiers for the packet have a fixed value for each packet instance. For this example parameters A and B will be set to 7 and 65 respectively. The counter value will increment starting at zero for each packet. The length of the packet will be calculated each time. The time stamp for the packet will be the system time for the sending system and is reset each time a packet is sent. Finally, the checksum will change every packet based on the values of the other parameters and is calculated by TReK. Table 6 shows an example of the first three packets generated and each packet attribute value.

	ID A	ID B	Counter	Length	Time Stamp	Checksum
1st Packet	7	65	0	100	2014-04-22 15:38:05	0xab31
2nd Packet	7	65	1	120	2014-04-22 15:38:10	0x1e49
3rd Packet	7	65	2	114	2014-04-22 15:38:15	0xf76b

Table 6 Packet Attribute Values

7 Calibration

Additional processing of parameter values is available and referred to as calibration. There are two built in types of calibration for TReK: polynomial and spline. In addition to the built in calibration types, you can perform unique calibration in your own code. Each of the calibration types is defined in the following sections.

7.1 Polynomial Calibration

Polynomial calibration uses a polynomial equation of any degree to calculate the calibrated value based on the input (converted value). The equation below shows the generic form of an nth order polynomial:

$$y = C_n x^n + C_{n-1} x^{n-1} + \ldots + C_1 x + C_0$$

Where y is the calibrated value, x is the converted value, n is the order of the polynomial, and C are constant values for each term.

7.2 Spline Calibration

Spline calibration performs calibration of a series of line segments as shown in Figure 8. The calibrated value is found by identify the line segment which contains the converted value of the parameter and using linear interpolation.

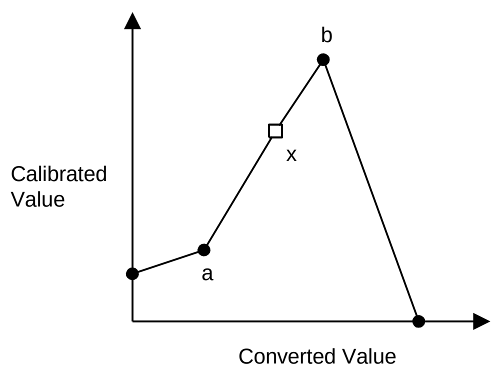

Figure 8 Spline Calibration

The calibrated value of a parameter is found using the following equation:

$$cal_x = (cal_b - cal_a)(conv_x - conv_a) / (conv_b - conv_a) + cal_a$$

7.3 User-Defined Calibration

The built in calibration types for TReK are sometimes not sufficient to calibrate a parameter. For those cases, TReK has introduced the concept of user-defined calibration.

User-defined calibration is code that you write to perform the needed mathematical functions needed to transform the converted value to a calibrated value. TReK will call the code you write whenever a parameter's calibrated value is needed. There is more information about user-defined calibration available in the Data API online help.

7.4 Calibration Example

In a previous section the raw and converted values for a binary code decimal were shown. The converted value (4791) can be calibrated with any of the above calibration types. For this example, we'll use the simple polynomial equation:

$$y = 0.0015x^2 – 7x + 3$$
$$y = 0.0015 (4791) – 7 (4791) + 3$$
$$y = 896.5215$$